Deep Reinforcement Learning for Simulated Autonomous Driving

Adithya Ganesh, Joe Charalal, Matthew Das Sarma, Nancy Xu
CS 229 Final Project

Abstract

This research applies deep Q-learning to train an agent that can autonomously drive in the Open Racing Car Simulator (TORCS) [1, 2]. An initial proof-of-concept algorithm was implemented in Flappy Bird. We compare the performance of classical reinforcement learning models, as well as deep Q-learning with fully-connected and recurrent neural networks.

Methods

As a proof of concept, initial experiments applied classical Q-learning to a discretization of Flappy Bird. We characterized the Flappy Bird state by the vertical and horizontal separation of the bird and the next obstacle and the vertical velocity of the bird. We compress this state into \( S \subseteq \mathbb{R}^2 \) with \( |S| \leq 50, 0 \leq \Delta x \leq 40, |v_y| \leq 9 \). Our action is a binary decision whether to jump or not. We reward staying alive and receiving a reward of \( R = 1 \) and penalizing with a reward \( R = -10000 \).

Background and Motivation

The Open Racing Car Simulator (TORCS) is a modern simulation platform used for research in machine learning and autonomous driving. Training an autonomous driving system in simulation offers a number of advantages, including cost-effective training. Many autonomous driving systems are trained in an open-source simulation platform used for research in machine learning and autonomous driving. Training an autonomous driving system in simulation offers a number of advantages, including cost-effective training for autonomous driving. Training an autonomous driving system in simulation offers a number of advantages, including cost-effective training for autonomous driving.

Deep Q-Learning Objective Function

Consider a deterministic policy \( \mu : S \rightarrow A \) parameterized by \( \theta \in \mathbb{R}^l \). Let \( r_t, \delta_t \) denote the total discounted reward from time-step \( t \) onwards:

\[
    r_t = \sum_{k=t}^{\infty} \gamma^k \delta_t(\pi_t q_k),
\]

where \( 0 < \gamma < 1 \). We can define a performance objective \( J(\mu) \) as the total discounted reward for a given policy; while the Q-function calculates the total discounted reward given a starting state-action pair:

\[
    J(\mu) = \mathbb{E}_s(\nu(\mu,s,A)|\mu)\cdot Q^*(s,A) = \mathbb{E}_s(\nu(\mu,s,A)|\mu)\cdot \mathbb{E}_s(\nu(\mu,s,A)|\mu)\cdot \mathbb{E}_s(\nu(\mu,s,A)|\mu).
\]

To dimunish the effect of highly correlated training data, we train the neural networks with a large replay buffer \( D \) that accumulates 100,000 samples. During training, we update our parameters using samples of experience \((s,a,r,s') \sim U(D)\), drawn uniformly at random.

\[
    L = \mathbb{E}_{s,a,r,s'}(U(D)) \left( r + \gamma \max_{s',a} Q'(s',a) - Q(s,a) \right)^2.
\]

Using nonlinear functions to approximate the Q-function often yields instability or divergence [7]. Here we exploit experience replay [3] to increase efficiency and mitigate instability by smoothing over changes in the data distribution. When training RNN models we sample blocks of training data from the same episode with length equal to the buffer network depth. To encourage sensible exploration, we use an Ornstein-Uhlenbeck process [5, 8] to simulate Brownian-motion-affected cars with respect to its momentum. This policy avoids pathologies of other exploration policies in autonomous driving that frequently cause the car to brake and lose momentum.

Neural Network Architectures

Architecture A: Fully-Connected Networks. In our actor-critic approach to learning, the critic model estimates Q-function values while the actor selects actions based on those estimates. The critic model learns to map time-series trajectories to Q-function values. We use deep Q-learning with fully-connected and recurrent neural networks. Our critic model is an efficient way to simulate model architectures for autonomous driving. Training a neural network for autonomous driving is an efficient way to simulate model architectures for autonomous driving.

Architecture B: Recurrent LSTM Networks. A powerful variation on feedforward neural networks is the recurrent neural network (RNN). In this architecture, connections between units form a directed cycle. This allows the network to model dynamic temporal and spatial dependencies from time-series data. Long short term memory (LSTM) networks are a robust variant of RNNs that capture long term dependencies between states. We have implemented an LSTM network using Tensorflow and Keras, and are currently evaluating the driving performance of different architectures. The LSTM cell can be mathematically formulated as:

\[
    f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \quad C_t = f_t \cdot C_{t-1} + i_t \cdot \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \quad o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \quad h_t = o_t \cdot \tanh(C_t).
\]

Training Models

By modeling long-term state dependencies, we expect LSTM networks to be capable of learning superior policies. We are currently using the RNN hyperparameters to achieve optimal convergence. In the future, we would like to test a variety of reward functions, exploration policies, and adaptive gradient descent strategies to optimize the likelihood of near-optimal and rapid convergence. Additionally, we would like to explore the performance of transfer learning to physical hardware and real-world control systems.

Conclusion and Future Work

We have demonstrated a deep Q-network that can autonomously drive in TORCS, with robustness over diverse environments (including the Aalborg, Alpine-1, and A-Speedway TORCS tracks). Applying reinforcement learning to train an autonomous driving system is an efficient way to simulate model architectures and fail safe modes without expensive labeling effort and physical hardware. By modeling long-term state dependencies, we expect LSTM networks to be capable of learning superior policies. We are currently tuning the RNN hyperparameters to achieve optimal convergence. In the future, we would like to test a variety of reward functions, exploration policies, and adaptive gradient descent strategies to optimize the likelihood of near-optimal and rapid convergence. Additionally, we would like to explore the performance of transfer learning to physical hardware and real-world control systems.
References


